Introduction à Numpy

Renaud Costadoat
Lycée Dorian

Introduction

La bibliothèque NumPy (http://www.numpy.org/) permet d'effectuer des calculs numériques avec Python. Elle introduit une gestion facilitée des tableaux de nombres.

Il faut au départ importer le package numpy avec l'instruction suivante :

```
>>> import numpy as np
```

Certaines variables sont prédéfinies comme la variable π.

```
>>> np.pi
3.141592653589793
```

Tableaux et matrices

Le premier point fondamental de Numpy (et de Scipy) est la gestion des tableaux et des matrices grâce à la fonction objet `array()`.

```
>>> np.array([1,2,3,4,5,6])
array([1, 2, 3, 4, 5, 6])
>>> np.array([1,2,3,4,5,6],'d')
array([ 1.,  2.,  3.,  4.,  5.,  6.])
>>> np.array([1,2,3,4,5,6],'D')
array([ 1.+0.j,  2.+0.j,  3.+0.j,  4.+0.j,  5.+0.j,  6.+0.j])
```

Pour créer une matrice, vous pouvez utiliser `array()` avec des listes de listes Python.

```
>>> np.array([[0,1],[1,0]],'d')
array([[ 0.,  1.],
       [ 1.,  0.]])
```


Tableaux et matrices

Vous pouvez aussi générer des matrices vides (remplies de zéros) de tailles arbitraires.

```
np.zeros((3,3),'d')
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

Ainsi que des matrices identité avec la fonction `identity()`.

```
>>> np.identity(4,'d')
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

Opérations sur les matrices

Les objets matrice font ce qu'il faut lorsqu'ils sont multipliés par des scalaires :

```
>>> 0.125*np.identity(3,'d')
array([[ 0.125,  0.   ,  0.   ],
       [ 0.   ,  0.125,  0.   ],
       [ 0.   ,  0.   ,  0.125]])
```

Addition de deux matrices (sous réserve toutefois qu'elles soient de mêmes dimensions).

```
>>> np.identity(2,'d') + np.array([[1,1],[1,2]])
array([[ 2.,  1.],
       [ 1.,  3.]])
```


Opérations sur les matrices

Multiplication de matrice et produit scalaire.

```
>>> np.dot(np.identity(2),np.ones((2,2)))
array([[ 1.,  1.],
       [ 1.,  1.]])
>>> v = np.array([3,4],'d')
>>> np.sqrt(np.dot(v,v))
5.0
```

Produit vectoriel.

```
>>> u = np.array([1,2,3])
>>> v = np.array([4,5,6])
>>> np.cross(u,v)
array([-3,  6, -3])
```

Opérations sur les matrices

Multiplication de matrices membres à membres

```
>>> a=np.array([1,2,3])
>>> b=np.array([3,2,1])
>>> np.multiply(a,b)
array([3, 4, 3])
```


Opérations sur les matrices

La fonction `shape` permet de déterminer la dimension d'une matrice.

```
>>> m = np.array([[1,2],[3,4]])
>>> m.shape
(2,2)
```

Les fonctions `determinant()` , `inverse()` et `transpose()` produisent les résultats attendus. Une transposition peut être abrégée en plaçant `.T` à la fin d'un nom d'objet matrice.

```
>>> m = np.array([[1,2],[3,4]])
>>> m.T
array([[1, 3],
       [2, 4]])
```

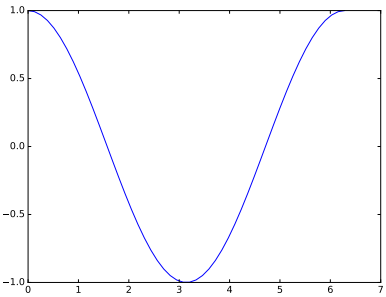
Espace linéaire, fonctions matricielles et tracés

La fonction objet `linspace(start, end, nb)` crée un tableau espace linéaire de points de valeurs comprises entre `start (inclus)` et `end (inclus)` en un nombre `nb` de points.

```
>>> np.linspace(0,1,20)
array([ 0.        ,  0.05263158,  0.10526316,  0.15789474,  0.21052632,
        0.26315789,  0.31578947,  0.36842105,  0.42105263,  0.47368421,
        0.52631579,  0.57894737,  0.63157895,  0.68421053,  0.73684211,
        0.78947368,  0.84210526,  0.89473684,  0.94736842,  1.        ])
```

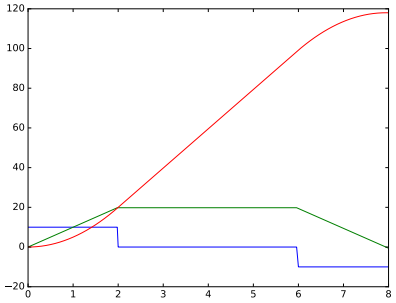
Cette fonction permet de tracer des données pour un graphique.

```
x = np.linspace(0,2*np.pi)
plt.plot(x,np.cos(x))
```



Espace linéaire: Exercice

Proposer le code permettant de tracer la figure suivante présentant, pour un solide en translation:



- l'accélération (en bleu),
- la vitesse (en vert),
- la position (en rouge).

Solveurs matriciels

Il est possible de résoudre des systèmes d'équations linéaires avec la fonction `solve()` .

$$\begin{cases} x+y+z=6 \\ 2*y+5*z=-4 \\ 2*x+5*y-z=27 \end{cases}$$

```
>>> A = np.array([[1,1,1],[0,2,5],[2,5,-1]])
>>> b = np.array([6,-4,27])
>>> np.linalg.solve(A,b)
array([ 5.,  3., -2.])
```


Solveurs matriciels

Plusieurs fonctions pour calculer des valeurs propres ainsi que des vecteurs propres :

- `eigvals()` retourne les valeurs propres d'une matrice
- `eigvalsh()` retourne les valeurs propres d'une matrice hermitienne,
- `eig()` retourne les valeurs propres et les vecteurs propres d'une matrice,
- `eigh()` retourne les valeurs propres et les vecteurs propres d'une matrice hermitienne.

```
>>> A = np.array([[13,-4],[-4,7]],'d')
>>> np.linalg.eigvalsh(A)
array([  5.,  15.])
>>> np.linalg.eigh(A)
(np.array([  5.,  15.]), np.array([[-0.4472136 , -0.89442719],
       [-0.89442719,  0.4472136 ]]))
```

Solveurs matriciels: Exercice

Proposer un code utilisant la bibliothèque Numpy permettant de résoudre le système d'équations suivant.

$$\begin{cases} x - 3 * y + 8 * z = -16 \\ 2 * x + 3 * y - 4 * z = 24 \\ 2 * y - 3 * z = 7 \\ 8 * x - 3 * z = 17 \end{cases}$$